# 目錄

# Web高阶学习

不是所有的鱼都生活在同一片海里。 每个人的世界都是如此不同。

泛游在知识的汪洋，你我迷惘而兴奋。信息过载的时代，海的尽头还是大海。

互联网的大潮汹涌袭来，其中包裹着一个忽隐忽现的见识：未来属于程序员？很多人想踏上编程这片红海，而苦于没有指南明针，无法辨别出市面上庞杂的书籍、网课、教程。本书的目的正在于此，为孤独的你提供网站开发最简洁的领域知识，帮你辨明方向。

我们不是不爱学习，很多时候只是不知道学什么。授人以鱼不如授人以渔，本书不仅包含 web 开发的方方面面，还会助你转变学习思维，带你用元认知能力提升学习效率。

向青草更青处漫溯。让我们扬帆起航！

> 学习不是一场战争。

这个世界上，最顶级的趋势叫时代。

大时代下，轰轰烈烈，小时代下，安分守己。幸运的是，我们正处在前所未有的信息革命大变局之下。

最顶级的趋势叫时代，次一级的是经济周期和行业状况。大数据、人工智能汹涌袭来，不甘平庸的你，可想要踏足前沿领域，站在浪潮之巅？

计算机领域知识庞杂， 学习编程，从开发一个网站开始，因为所有的应用都离不开 Web。欢迎你，来到这个诡谲的网络世界。

---

曾经我们以为学习像一场战争，在路途上我们攻克了一个个难题，打胜了一场场知识点歼灭战。

但这本书里我要带你使用另一个隐喻。学习不是一场战争，而是一棵树。

（Web 开发到底是难还是容易？）

Web 的内容太分散繁多，发展又太快，新技术不断出现，战争在这里往往失效，而树是兼具横向扩展与纵向扩展能力的最优雅的结构，也是人类的最佳抽象知识结构。在树的隐喻下你会明白，知识管理不重要，重要的是掌握生成知识体系的能力。

希望你从零开始，枝繁叶茂。

# 语言与框架

说到编程，我们首先想到的会是编程语言。我们熟悉那些排行榜上如日中天的编程语言，C、Java、Python、PHP，甚至 Lisp，就像对兵器谱上的兵器如数家珍一样，畅想着排行名次和对阵情景，有时还免不了争论一番哪个语言是最好的语言。

不过说到底，语言最重要的是用起来。你可能在专业课程上学习过 C++，学习过复杂的对象、指针、模板、设计模式，但似乎很少在项目中真正实践，尤其是 web 开发中。

这是因为，你作为一个学生在跟着老师或者课本学习基础知识时，你需要完全信赖，不带批判地学。而现在，当你真正开始实践项目，你需要做自己的老师，你来决定你要学什么。

尼采说，不加选择的知识冲动就像不分对象的性冲动一样，都是下流的标志。高明的学习者，就是文明的人类，用理性对抗本能。学习的第一理性就是要学会选择学什么。

选择学习哪种编程语言，选择哪一本书效果最好，选择学习哪个技术方向，选择投身哪个行业，这就叫做聪明，这也叫做见识。

一个技术能不能发展起来，关键看三点：

- 有没有一个比较好的社区。像 C、C++、Java、Python 和 JavaScript 的生态圈都是非常丰富和火爆的。尤其是有很多商业机构参与的社区那就更为人气爆棚了，比如 Linux 的社区。
- 有没有一个工业化的标准。像 C、C++、Java 都是有标准化组织的。尤其是 Java，其在架构上还搞出了像 J2EE 这样的企业级标准。
- 有没有一个或多个杀手级应用。C、C++ 和 Java 的杀手级应用不用多说了，就算是对于 PHP 这样还不能算是一个好的编程语言来说，因为是 Linux 时代的第一个杀手级解决方案 LAMP 中的关键技术，所以，也发展起来了。

具体到一种编程语言，除了这三点，还应考虑的是：

- 学习曲线是否低，上手是否快。这点非常重要，C++ 在这点上越做越不好了。
- 有没有一个不错的提高开发效率的开发框架。如：Java 的 Spring 框架，C++ 的 STL 等。
- 是否有一个或多个巨型的技术公司作为后盾。如：Java 和 Linux 后面的 IBM、Sun……
- 有没有解决软件开发中的痛点。如：Java 解决了 C 和 C++ 的内存管理问题。

就像上面这样，你通过一点点收集信息，不断拓展自己的视野，形成愈加全面的判断，生发出自己的知识和见识。

然后你又发现，选择语言时同时也是在选择框架。你搜索到目前主流的网站开发技术框架有：

- Python + Django
- PHP + Laravel/Yii/Yaf/...
- Ruby + Rails
- Java + ...
- Elixir + Phoenix
- Golang + Beego
- ……

慌乱地浏览社区里的大量讨论，你又了解到，选择框架还需考虑具体的项目，项目规模、项目周期、时间成本、人力成本……

信息汪洋一下子涌到你的面前。你迫切想要知道，选择的窍门是什么？

为什么有些人搜索信息的速度、质量都比一般人强？为什么有些人的信息品味优于一般人？知识无非就是信息，这个信息泛滥的时代，好知识何处去寻？

---

Tips：有一个 chrome 插件 wappalyzer 可以查看打开的网站使用了哪些技术，比较方便

# 技术

招聘网站公布的调查报告说，2017 年全国高校应届毕业生的平均薪酬大约是 5000 元。而入选教育部 42 所一流大学建设高校 (双一流大学) 的毕业生平均薪资大约是 6000 元。

差别并不大。然而，如果再展示出不同专业的数据会更有意思，比如，计算机专业的毕业生平均起薪可能会高得吓人。

越来越多人选择自学编程，越来越多培训机构出现，计算机科班出身的学生常有的一个疑问是，相比于他们，我有什么优势？

老实说，在编程技术上确实没什么优势。系统地研读理论，再开始实践，这是大多数领域的正确学习方法，然而在编程领域却失效了。编程的学习最大的“反常识”是，尽量少读理论，一开始就要实践。

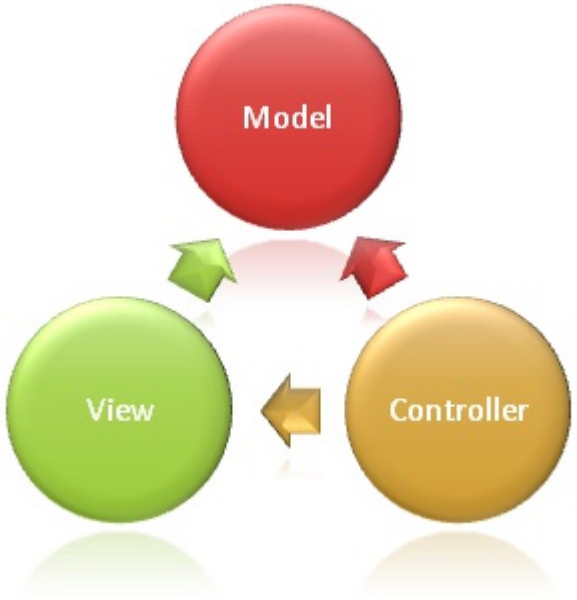
如前一节所述，项目开发的矛盾在于，所学与所用的脱钩。根本原因是技术发展日新月异，课堂所授知识难以跟上实际需求。

就软件开发来说，软件工程相关课程的理论很多停留在瀑布式、螺旋式开发模式，而实际的中小型项目时间成本难以为继，许多人推崇敏捷开发理念，倡导精益思想。这两者的对立，反映的正是专家与通才（全栈）的区别。

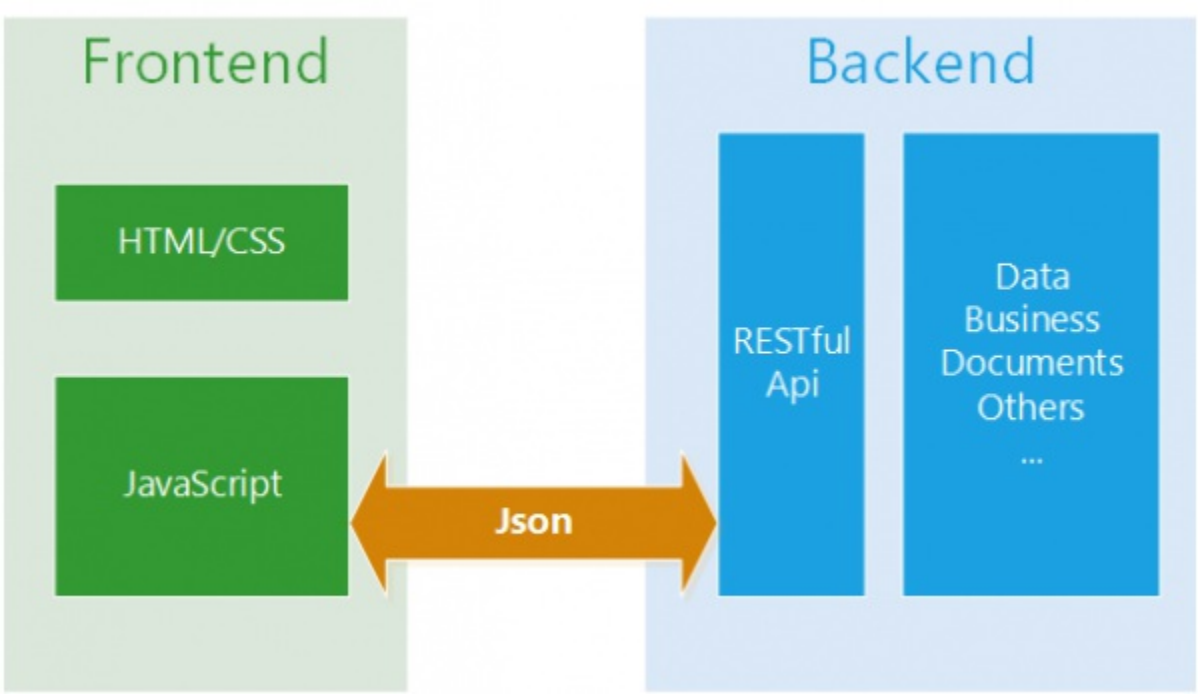
一方面，技术的革新使得软件开发的门槛下降得越来越快，另一方面，越来越多新技术的涌现要求个人掌握更多的技能。

举个例子，网站开发的历程，许多年前大多网站都是静态的，为了实现动态网站，人们使用 CGI 的模式，


直到后来 MVC 架构的出现，




才改变这种混乱的状态。而后，随着网站变得愈加复杂，同时伴随着大量工具的出现，模块化的趋势凸显，阻碍软件开发效率提升的一大因素便是沟通成本，慢慢地网站架构变为前后端分离的结构。




正是在实践中进行的检验，使得软件开发技术不断发展。行动第一，是学习编程最重要的道理。

无法接触业界实践，那么读计算机专业还有什么用处呢？读大学的用处又是什么？每当我有此疑问，我会想起马奇在斯坦福大学晚宴上的致辞：

> 高等教育是远见卓识，不是精打细算；是承诺，不是选择；学生不是顾客，是侍僧；教学不是个人工作，是圣事；研究不是投资，是见证。

可惜的是，今天的大学愈来愈像充斥利益的市场，早已不是象牙塔的形象，尤其是计算机专业的教育。当大学的目标变成培养更加适应社会的人才，摧残了年轻人的学习热情和内在动机，那也就不叫高等教育了。

就我的浅见，大学传授的不是最好的学习方法，而是如何学习的方法。编程最重要的是实践，而大学所提供的不是更接近公司的实际场景，大学传递学习理念：从实际中学习。

面对陌生的 Web 开发，你不知从何学起，不妨从一个网站如何运行开始。当你是一个普通用户，网站运行逻辑对于你是一个黑盒，你需要转变到一个开发者的视角，一步步摸清这个黑盒里面有哪些组件，你自然就明白需要哪些知识了。

同样的，你通过查看大公司的招聘网站，就明白了网站开发一般分为前端和后台两部分，毕竟所有公司都招聘这两种职位。再查看相应职位的招聘要求，你就知道了在实际中开发者最重要的能力是什么。

智者中的智者查理·芒格常说，反过来，反过来想，你会有意外的惊喜。这种从实际中学习的理念就是一种反过来想的逆向思维。你一定有过这样的感觉，当你突然掌握了一种思维方法，一种思维模式，那是和学到新的学习方法不一样的甜头。

你不由自主开始思考，高效学习者的“好模式”是什么？好知识的“好模式”是什么？

# Hello, World!

如果说好知识有什么“模式”的话，源头知识大于其他。

细心的你会发现，任何一种编程语言，都是从 Hello, World! 开始的，为什么呢？这来源于一个故事。

写 Hello, World! 的意义在于，运行第一个程序，入门这个语言。

关于如何快速入门，有太多的教程、文档、博客、书籍、论坛帖子在讨论了，选用哪一种可能是因人而异的，有些人习惯读书学习，有些人习惯听别人的指导学习，有些人喜欢实践（可能会去 Learn X in Y minutes 这样的网站，确实是有趣的好网站）中学习。

但最适用于所有人的好方法是，从官方文档学习，因为这是源头。并且，绝大多数优秀语言都有优质的官方文档。

例如学习 PHP 这种难以出错的简易语言，到 PHP Manual 上快速学过一遍最常用的 string 和 array 就可以上手了，顺便再把 PHP 之道 翻一翻更佳。另外 Yii 也有详细的官方文档。文档有可能难读，那么还有官方认可的社区，是另一源头。

事实就是这样，最重要的知识，最有效的知识，就在屈指可数的那些地方放着，只是多数人偏偏绕道而行。许多程序员写程序时遇到 bug，直接复制错误信息粘贴到百度搜索栏里，然后挨个条目打开，各色网页里各种字号的各式各样的解答让人眼花缭乱，逐个看一遍往往还发现很多内容是完全一样的。其实这些解答的源头基本全都是 StackOverflow（这是全球最大的程序员问答社区），为什么不直接去 Stackoverflow 找呢？还有些问题源头是 GitHub 的 issue，那是开源代码作者解决 bug 的地方，没有人比作者更清楚错误原因了吧？

不过，即使知道了这些，很多人还是觉得难。论文当然比书超前，可是谁愿意读密密麻麻的艰涩论文呢？谁又愿意用英文慢慢啃枯燥的官方文档呢（其实现在很多文档都有了中文翻译版）？当不想做一件事时，人会找出无数的理由。

信息过载，时间又变得碎片化，人们每天被动地接受铺天盖地的信息轰炸，看一本书时总想拿出手机刷一会儿，手机上看一篇文章，又总被新消息打扰，阅读能力直线下降。 哪里还能踏踏实实读文档？

再者，信息迅速涌现，制造出一个个信息茧房。有人每天泡在贴吧里，有人整天沉浸在微博里，一直关注同一个话题，软件算法又会过拟合推送出更多相似内容，于是信息流里每天都是同样的东西变个花样出现。同样地，学习知识也不能沉浸在小世界中，那就像是坐井观天，永远无法开阔视野，增长见识。

慢慢地你会明白，承载信息的人大于信息。假如你要学习某样东西，学习资料多得你永远看不完，而你只关注某个最厉害的牛人，看他的学习路径，或者加入最好的社群，那里面就是最好的信息。面对人，而不是面对信息，你会发现原来信息并不重要，信息承载的人才重要，如此你就摆脱了信息焦虑。

# 善用工具

好的工具层出不穷，涉及到项目开发的各个阶段，都有相应工具提升自动化程度，让你从繁重的重复性工作中解放出来。很多情况下，我们只是不知道从哪里挑选工具而已。

哪里有需求哪里就有市场，在互联网这个开放的世界里更是如此。

一开始人们需要写程序，于是有了编辑器，接着有了更多编辑器，接着有了编辑器的对比评测，有了编辑器聚合工具，接着有了如何选工具的工具。

你感到乱花渐欲迷人眼，但你一定要知道的是，工具只是辅助。

> “好的装备确实能带来一些帮助，但事实是，你的演奏水平是由你自己的手指决定的。”
> –《REWORK》

工具是为了提升效率，追求效率才是我们的根本诉求。这时你会明白，小而美的工具可能不够实用，大而全的工具可能不够简洁，不存在完美的工具，一切都要看场景。

刚入门编程时你写一个输出一段文本的程序，可能 IDE 还没有打开在 vim 里就已经写完了；后来你写一个上千行的程序，在 vim 里改来改去也运行结果不对，但在 IDE 里 debug 一次就顺利找到错误原因了。那么是 vim 好还是 IDE 好呢？

我在做项目时，一开始用的是 Sublime 和 Atom 这样的轻量编辑器，之后项目代码逐渐增多，渐渐难以把控，我换上了 WebStorm，文件跳转、运行调试、版本管理一下子方便了许多。可见，不同场景不同阶段需要不同工具。

不过对一些新手来说，有时候是不知道有哪些工具可供选择，这就需要学会有效的搜索（关于搜索其实有多方面的技巧，这里指的是基本的搜索引擎和问答社区等地的搜索），以及结合别人的推荐，便可以了解到绝大部分的工具。

- 许多优质的工具都是开源的，开发者永远要记得 GitHub
- 有一些良心的工具聚合平台，比如小荷尖
- 还需要多发现最新的应用，最好的地方在 36kr Next / product hunt

这时你已经拥有了一堆工具，但别忘了那句古老的箴言：

> 如果你是锤子，那你眼里的一切都是钉子。

> 全局把握。

星辰起落，沧海桑田，古老的民族在刀耕火种的徐徐演化中走到今天，知识大爆炸使得农耕文明下的人们无所适从，我们要做知识的游牧民族，哪里水草丰美，就迁徙到哪里繁衍生息。

不再有哪个突破不是靠着多学科多领域的交叉，每个人都需要掌握新的知识。只是知识这颗水草，该怎样有效汲取养分？

MIT 计算认知科学实验室主任 Josh 比较了循环结构、方块结构、星形结构等抽象知识形式，最终证明知识的最佳抽象结构是树形结构。每天使用的计算机文件夹就是树形结构，记忆、存储、组织、内化，人的大脑在不知不觉间习惯使用树整理知识。

对于树，最常用的有效搜索方式有两种：广度优先搜索，深度优先搜索。

人类对树形结构早已熟稔于胸，总结出两种有效的搜索方式：广度优先搜索，深度优先搜索。前一章已经论述过寻找好知识的广度优先搜索方法，然而善于脑补和偷懒的人类大脑却是排斥深度学习的。

人类大脑天性如此，读一句话时都在脑补，“哪怕这话句的序顺不对”，斯坦诺维奇详细论证了这一点——大脑是个吝啬鬼。深度学习违背了大脑的特点，所以人很难掌握。

《哈佛商业评论》上，有一个「大师的大师」排行榜，其中仅次于彼得·德鲁克的，是低调的詹姆斯·马奇。

马奇的著述不多，但本本经典，有一本演讲整理的册子《经验的疆界》很薄，却是探究本质的好书。

你可能已经有所体会，编程非常依赖模仿他人和实践经验，很多时候你不需弄懂，只需依据经验照抄一下就能完成目标，但又担心没有学会。马奇就针对这个问题作了深刻剖析。

所谓学习，就是在观察行动与结果联系的基础上改变行动或行动规则。如果那些改变是改进，那么学习就促进智慧增长。学习经常而且容易带来改变，但是学习不一定促进智慧增长。

高智学习一定是在理解知识因果结构基础上的。这往往需要溯源，然后建立全局的认识。我建议的学习思维是，用学习一门学科的方式学习网站开发。

如何构建一门学科？站在知识源头，追溯学科提出的核心问题。建立核心话语系统，学科内用何种概念、语言、结构，探索核心问题。比如，社会学系下的社会心理学与心理学系里下的社会心理学，话风不同。佐以二级推演体系，大量证据，补充核心话语系统中的抽

象概念。警惕知识的诅咒，切勿手里一把锤，万物皆成钉，要试着从其他学科视角，思考问题。

一门学科总是在三个方面徐徐展开，核心问题和话语体系谓之模型，历史发展和人物情况谓之故事，行为改变和习惯养成谓之行动。请记住模型、故事、行动三个关键词。

演绎而非归纳。

# 优雅环境

搭建一套优雅的开发环境，需要经过诸多尝试，这个过程既让人兴奋，又让人烦躁，因为选择太多，细节太多，每个人的品味也是多样。

本节我只列出我的环境配置，尽可能全面，而不详述筛选和对比不同环境的优劣。除开硬件，从操作系统开始，分为 MacOS 和 Windows 两篇。

## MacOS

对开发者来说，OSX > Linux > Windows，不需解释。

如果你已经有了 Mac，恭喜，让我们开始挑选趁手的兵器吧。

Mac 上自带最好用的全文检索和快速启动功能——Spotlight，必备，如果进阶玩家还想要更多花样，请选择 Alfred，用过都说好。

程序员最离不开的系统自带软件——Terminal（终端），（OSX 和 Linux 相对 Windows 最重要的一个优势就是对 shell 的强力支持）实际上在终端里可以完成绝大部分工作，值得好好自定义成用起来更舒服的终端，我使用 iTerm 和 Oh My Zsh。

Mac 上预安装了 Java、Python、PHP 、Git 等，非常方便。安装其他语言、工具时，请选用 Homebrew，包管理工具太好用，必备。

我写 PHP，使用的 XAMPP 集成安装，省事。

为了使用 SVN，安装了 Cornorstone。

Web 开发离不开网络调试，安装 Charles 不会多余。

写代码我使用 Sublime 和 PHPStorm。

虽然 Safari 很不错，但开发还是用 Chrome / Firefox。

## Windows

很多时候不得不用 Windows，花些时间找到相应的替代软件也没问题。

自启动软件用 Wox，替代 Alfred。全文检索用 Everything，替代 Spotlight（虽然大大不如，勉强可用）。

终端用 CygWin（并不是很满意），Powershell 也比较常用。

软件包管理用 Chocolatey。

当然，Windows 也有许多优势，毕竟用户量大，软件可选项多。

需要安装 Git 和 SVN。

Sublime 和 PHPStorm 也有安装。

数据库管理用 Navicat。

鉴于终端不好用，再装个 Xshell 吧。

# 协作软件

软件开发中影响进度的核心因素是沟通。

从大公司开发大型软件，到更多小团队快速开发软件，软件开发的发展趋势便是不断消除这一矛盾。

正因为此，出现了越来越多的开发协作工具。本节将要列出一些优秀的团队协作软件以供参考，而避开团队管理中诸多问题的讨论，那些实际上比工具更为重要，但已经属于管理学的范畴。

---

## platform of products posts

product hunt

NEXT

## Teamwork

国外

- Slack
- Trello

国内

- Tower
- Workfile
- 零信

---

Teamwork工具总体上解决两方面的问题：沟通和协作。

### 沟通

沟通包括：IM、Notification、Personal exchange。

- Instant Message有的在工具内实现，有的集成其他IM工具如钉钉、企业微信
- Notification指各种第三方服务的通知
- Personal exchange是一些私人化的动作，如Incoming webhooks、Bots、Command、Outgoing webhooks

## 协作

协作包含任务管理、计划实施、进度安排、文件共享等所有推进项目进展的工作

国内协作工具各有特色，选择时主要从价格和用户习惯方面考虑

---

协作与沟通整合的方案：钉钉+Tower、Tower+零信、Slack+Trello

# 版本控制

版本控制是写代码时必不可少的一个环节。

> 版本控制（**Revision control**）是维护工程蓝图的标准作法，能追踪工程蓝图从诞生一直到定案的过程。此外，版本控制也是一种软件工程技巧，借此能在软件开发的过程中，确保由不同人所编辑的同一代码文件案都得到同步。

## Git 与 SVN

有两种类型的版本控制软件，中央式系统和分布式系统，两者的代表分别是 Git 与 SVN。关于两者利弊的讨论有一大堆，可以全部忽略掉，直接选用 Git，原因很简单，Git 在实际中使用越来越频繁，用途也更加广泛，不光编码，写论文、写文章、写书都可以方便地使用 Git，而 SVN 只见于多年前的大型软件系统。

> 分布式系统 Linux 内核的发明人林纳斯·托瓦兹就是分布式版本控制系统的支持者，他开发了目前被开源社区广泛使用的分布式版本控制系统 Git。

## Git 命令

虽然每个操作系统都有 Git 图形化操作客户端，但仍然建议学习 Git 命令，使用几次熟练后，效率会大大提升。最常用的命令无非以下几个：

```
git clone
git add .
git commit
git push
git pull
git reset
git branch
```

快速掌握 Git 可以参考官方文档和廖雪峰的教程 。

# 工作流

# 前端

## 发展历史

先来看看网页的发展史。

提到前端时，你可能想到的是 HTML、CSS，没错，最早的前端页面就是从 HTML 开始的。超文本标记语言（HyperText Markup Language）简称 HTML，一个 html 文件就是一个静态网页，把这个文件放到 web 服务器上，并在服务器上配置好路由规则，那么当浏览器请求某个 URL 时，web 服务器就会把对应的 html 文件扔给浏览器，再经过渲染，就成了网页。

但是网络上的网页需要有区分，比如面向不同用户要展示不同内容，html 文件不能简单扔给用户，需要有一些逻辑操作。这个任务最初是由 C、C++ 这些语言承担的，服务器操作完后向浏览器扔过去拼接后的字符串，替代了 html 文件，这就是 CGI：Common Gateway Interface。

然而，随着网页内容越来越多，字符串难以完成任务，于是出现了动态 html，也就是在 html 文件中加入了 <%=var%> 这样的变量，这种方案有三种方式可以实现：ASP、JSP、PHP。其中，ASP 对应的是微软的 .NET 技术，JSP 对应的是 java 的方案，PHP 则是开源社区开发的。

但这种动态还不够动态，人们还想动态更新页面内容，更新时只能再请求一次服务器发一份新的 html 文件，这显然不够高效。直到 JavaScript 的出现，完全改变了网站的发展（顺便说一句，JavaScript 与 java 并没有任何关系）。有了 JavaScript，便可以通过修改 html 的 DOM 结构和 CSS 来实现更丰富的网页，不过这些动作都是在浏览器而不是服务器完成的，我猜这时候才慢慢有了前后端的概念吧。

不过很快人们发现，写一大堆 JavaScript 挺麻烦的，使用浏览器的原生 API 操作 DOM 很繁琐，而且经常出错，这时出现了 jQuery，大大简化了代码，终于使得前端开发变得容易。

同时，后端也没闲着，慢慢地发展出高效的模式，借鉴桌面应用程序的 MVC 思想，网站开发也有了经典的 MVC 模式，前后端配合得很默契。之后，网站应用越来越复杂，用户对于交互性的要求越来越高，单纯的 MVC 已经难以胜任，经过改进，MVVM 呼之欲出。

## 技能

前端技术更新很快，每一年都有很多新东西出来，也会有些淘汰的东西，从五花八门的前端框架就能看出来了。前端开发者需要掌握的核心技能包括（最好按顺序学习）：

1. Uniform Resource Locators (aka URLs)
2. Hypertext Transfer Protocol (aka HTTP)
3. Hyper Text Markup Language (aka HTML)
4. Cascading Style Sheets (aka CSS)
5. JavaScript Programming Language (aka ECMAScript 262)
6. JavaScript Object Notation (aka JSON)
7. Document Object Model (aka DOM)
8. Web APIs (aka HTML5 and friends or Browser APIs)

下面列出一些相关的链接

- HTML 5.2 from W3C
- HTML attribute reference
- HTML element reference
- CSS reference
- W3C DOM4
- ECMAScript® 2017 Language Specification
- Web API Interfaces
- HTTP/2
- Introducing JSON
- JSON API

## 框架

几个流行的前端框架

## 基于 **Jquery**

入门简单，API 友好，使用广泛，受 MVVM 影响近几年有所没落

## 基于 **Angular**

组件多，体验好，风格一致

## 基于 **React**

强大，复杂交互都能实现，开发成本低。入门较难，性能不高

## 基于 **Vue**

美观，火爆，设计新颖，组件不够多

# 后端

后端处理的问题多而杂，并且，这些问题还一直在变化！没办法，程序员是一个需要不断学习的职业。

## 语言

先从语言说起，大部分的编程语言都可以做网站后台开发。

- C / C++，基础中的基础，对初学者不友好，但在许多系统中异常重要。
- Java，成熟的 J2EE 使得 Java 成为许多大型网站的首选。
- PHP，据说是“世界上最好的语言”，这个梗我一直没明白出处何在，大概是因为用户多吧。
- Python，简洁，划算。
- Go，专为网站开发而生，性能不错。

## 数据库

网站是离不开数据库的，总得存点东西吧。最常用的开源数据库 MySQL 足以应付小型网站，一般要考虑的问题是，如何持久化连接数据库？这么普遍的需求大部分语言都有成熟的解决方案。

提到数据库设计，人们熟知的是数据库概念结构设计、逻辑结构设计、物理结构设计，这些必要知识是没错的，但往往成为初学者的学习负担，设计数据库怎么这么复杂？费这么大劲设计好又怎么实现呢？毫无头绪。数据库的设计当然复杂，只是初学者更关心的是怎样把数据库使用起来。你可能需要的是 jdbc、PDO 这样的接口，能够通畅地访问数据库。不过，等到业务更加复杂，又会发现离不开基础的设计和优化了。

……关系型数据库，非关系型数据库……

## 软件架构

……MVC、MVP、MVVM……

## 服务化

# 云&大数据

网络搭好之后可还远远没有技术，随着网络应用越来越复杂，需要更多优质的基础服务提供保障，相比于自己搭建这些基础服务，云服务厂商提供的服务在稳定性和扩展性方面要可靠得多。


这些服务不仅包括数据库、域名服务器、文件存储等基础组件，还有 CDN、负载均衡、日志、NAS、大数据分析、搜索组件等等。

# Dev&Ops

> Talk is cheap, show me the code.

编程最重要的还是实践，这是一个共识，但为什么是这样？实践给你带来了什么呢？在实践中如何学习和创新？

一位享誉世界的日本学者的研究可以加深我们对创新的理解。在野中郁次郎看来

> 人类的知识分为两部分，一部分是形式知识，它可以用正式的语言表述，包括语法陈述、数学表达式、技术规范、手册等。因此，这类知识可以在个体之间正式地且方便地进行传播。然而，除此之外，还有另外一类难以用正式语言表述的知识——暗默知识——更为重要。它属于植根于个体经验的个人知识，涉及像个人信念、视角及价值体系之类的无形要素。

最小行动。第二序改变

学习，使用树的隐喻，而非战争隐喻

使用演绎的方式，而非归纳

马奇

所谓学习，就是在观察行动与结果联系的基础上改变行动或行动规则。如果那些改变是改进，那么学习就促进智慧增长。学习经常而且容易带来改变，但是学习不一定促进智慧增长。

低智学习，基础是复制成功，最常见的是复制给你带来好处的规则和行为，远离给你带来失败的规则和行为。试误、模仿、天择是低智学习的三种机制这种学习、人会在，动物会做，就连植物也会依据环境做出相应的反应。

另外一方面，高智学习最常在人类中出现，只有人类才会深究因果，总结经验，用文字、符号加以记录，进行传播。

程序员最重要的能力是专注。

# 编码习惯

开始写代码了，撸起袖子加油干吧！可没有这么简单。

优秀开发者追求的永远是效率，而不是工作量，更不是喊着口号浪费时间和资源。

当多名程序员在多个项目中合作时，就需要一个共同的编码规范，这种规范既要求事无巨细的精确，又要求适当的灵活，谓之风格。

遵循统一的良好风格，程序员在协同时将避免无益的争端，大大提升工作效率。同时，恪守良好编码风格对程序员个人大有裨益，这就像初学语言练好发音、初学写作练好字一样重要，举重若轻的背后是良好的习惯养成。

幸运的是，好的编码规范是所有优秀程序员的共识，github 上有开源出来的详细规范标准。例如，我在使用 PHP 语言时遵循的规范来自PHP-FIG翻译。 好的编码规范会细致到变量命名、行首空格、对齐方式等等容易忽略的细微之处，但一定是必要的。

不过这还不够，团队协作时不仅要共同维护同一份代码，还有同一份文档，以及共同使用工具。

GitHub 不仅仅是一个远程仓库，github 本身就是一种编码规范。新建仓库写 Readme，发 issue 提问题，pull request 做贡献，是这一套模式成就了最活跃最全能的线上代码社区，反过来，这个社区鼓励每个人遵守规范，养成良好开发习惯。还记得知识的分类吗？GitHub中有大量的内隐知识，是一个宝库。

# 行动

行动 > 目标

执行意图

第三个建议是关于树立目标。报了开智学堂编程班，很多同学可能在自己的小本子上偷偷写上一句话：「我要在三个月内学会编程」。虽然说不是 21 天学会编程吧，但很多同学都以为三个月能学会编程。然而，一旦写了这个目标，绝大多数人最终的结果其实是学不会编程，完成不了这个目标，为什么呢？

这是认知科学研究中很有意思的现象。因为人类大脑特别喜欢脑补，一旦把「我要在三个月内学会编程」这个目标写下来，你的大脑会怎么去处理？

它会认为这事情已经完成了。就像「新年我要成为更牛的人，要去周游世界，我要 XXX」，因为没有提出一些具体的指令，你的大脑会把这些目标扔到「完成区域」。在未来的三个月内，你的大脑不会有任何行动。这是很不好的制订目标的习惯。

认知科学家格尔维茨（Peter Gollwitzer） 把「我要三个月内学会编程」这样制订目标的习惯称为「目标意图」。与目标意图相反的是「执行意图」。它是这样一种方式，采取的是如果...那么...的句式。就是说它把你更具体的指令下达给你的大脑。「如果每天晚上八点之前回到宿舍，我立即打开笔记本开始登录 GitHub 网站、提交自己的代码」，这是一种新的制定目标的方式：执行意图。这种方式有何不同？

- 第一它包括时间因素：「晚上八点」；
- 第二它包括地点因素：「回到宿舍」；
- 第三它包括具体可操作的事情：「提交代码」。

这样一来，你在上周给自己大脑布置了这样任务，然后很忙，把这个事抛在脑后，但是大家都很明白了，人类大脑喜欢脑补，到了下周晚上八点的时候，你的潜意识就会自动进行脑补，现在晚上八点了，我已经在宿舍了，这个时候我要干什么 —— 写代码。

认知科学家研究发现通过改变一种目标制订的方式，学习效率提高了三倍以上。格尔维茨的妻子，另一位心理学家 Gabriele Oettingen 则在执行意图基础上，结合心理对照，提出了 WOOP。这是任何一位开智学堂的学员入门第一课。

用执行意图的方式制订目标。

如果……就……

当你还是计划阶段是，你倾向于关注为什么，此时需要补充思考可行性，一旦开始执行，应更多考虑是什么。

人类习惯将遥远的目标采取抽象化处理，忘记可行性；一旦开始执行较难的目标时，则又开始纠结意义。

因为人是喜欢挠痒痒的猴子，这是人类社会进化的一个规律。

人类最早的学习是学徒制，知识就是八卦与故事。

当你抓住自己的头发，你是打不死自己的，你下不了手。但是让你的小伙伴抓住你的头发，你会感觉到很痛苦。

「即使你只是在头脑中想象如何教别人，这时候你的学习效率也是自己学习效率的两三倍以上。」

主动去做那一只帮帮人挠痒痒的猴子。